$1.80

Apple Assembly Line

Volume 4 -- Issue 12 September, 1984

In This Issue...

News about Micromation.

Jack Lewis's company, which among other things makes a line of Apple-related products to support the Heathkit Hero robot, has changed its name to Arctec Systems, Inc.

Jack also has a stand alone voice recognition system with an RS-232C interface which may be of interest to some of you. It contains a 65C02 processor, 4K ROM, and 16K of battery-backed-up RAM. Speaker-dependent recognition of up to 256 words or short phrases is possible, with 95-98% accuracy claimed. Arctec's number is (301) 730-1237, in Columbia, Maryland.

And Some Bad Tidings

The saddest news I have heard lately is of the demise (bankruptcy) of Softalk Publishing. Softalk has been my favorite of all the magazines devoted to the Apple. At this point I do not know how to obtain copies of any of their back issues, or of the books they have published. I assume, and hope, they will be available again soon. With the passing of so many companies, via Chapter 11, many magazines are having great difficulty this year. Unpaid advertising bills then cause a domino effect....

18-Digit Arithmetic, Part 5..............................Bob Sander-Cederlof

There is a lot of ground to cover in this installment, so I have been forced to use smaller type to squeeze it all in. I want to describe and list the code for the linkage to Applesoft, and for handling arithmetic expressions.

Loading and Linking to Applesoft

The ampersand (&) statement, according to the Applesoft Reference Manual (page 123, top of page) is

> "intended for the computer's internal use only; it is not a proper Applesoft command. This symbol, when executed as an instruction, causes an unconditional jump to location $3F5. Use reset ctrl-C return to recover."

Not so! The &-statement is intended for adding extensions to the Applesoft language! It does cause a jump by the Applesoft interpreter to $3F5. If you have not set up any extensions you will get a syntax error when you use "&". But if you have extensions installed, you can work all manner of miracles. DP18 is one such miraculous extension. There are many more around, both in the public domain and in the form of commercial products.

This of course leads to a problem. What if you want to use two or more such extensions? I have written DP18 so that you can chain together one or more additional extension packages as you see fit.

It is very important to decide where the DP18 package will reside in memory. I spent weeks tossing around various options, back when I was designing the DPFP 21-digit package. Of course, at that time, Apples came equipped with anywhere from 16K to 64K RAM; now you can depend on almost all Apples having at least 48K RAM. I still favor the decision I made four years ago, to load the double precision code at $803, after shifting the Applesoft program far enough up in RAM to leave room.

I have a program I call ML LOADER, which is included on the S-C Macro Assembler disk as a sample program. It performs the function of moving an already-executing Applesoft program up higher in RAM. By including the following line at the beginning of my Applesoft program, I can load DP18 and link it to the & hook at $3F5:

```
10 IF PEEK(104)=8 THEN PRINT CHR$(4)"BLOADB.ML LOADER"
  :POKE 768,0 : POKE769,30 : CALL770
  :PRINT CHR$(4)"BLOAD DP18"
  :POKE 1014,PEEK(2051) : POKE 1015,PEEK(2052)
```

PEEK(104) looks at the high byte of the starting address of the Applesoft program. Normally Applesoft programs begin at $801, so PEEK(104)=8. If DP18 has not yet been loaded, then PEEK(104) will still be equal to 8. If it has already been loaded, then the rest of line 10 is skipped.

B.ML LOADER loads at $300. Its function is to shove the Applesoft program higher in RAM. You POKE the distance to shove into 768 (low byte) and 769 (high byte), than CALL 770. When you wake up an instant later, you have been relocated. The Applesoft program keeps on executing as though nothing happened. Only now there is a gaping hole between $800 and whatever.

```
S-C Macro Assembler Version 1.0.....................................$80
S-C Macro Assembler Version 1.1..................................$92.50
Version 1.1 Update...............................................$12.50
Source Code for Version 1.1 (on two disk sides)...................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code).....................$20
S-C Cross Reference Utility (with complete source code)...............$50
DISASM Dis-Assembler (RAK-Ware)......................................$30
Source Code for DISASM.................................additional $30

S-C Word Processor (with complete source code).......................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk..............................$15

(All source code is formatted for S-C Macro Assembler Version 1.1.  Other
assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.........................................each $15
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1:  Oct-Dec 1980    QD#2:  Jan-Mar 1981    QD#3:  Apr-Jun 1981
    QD#4:  Jul-Sep 1981    QD#5:  Oct-Dec 1981    QD#6:  Jan-Mar 1982
    QD#7:  Apr-Jun 1982    QD#8:  Jul-Sep 1982    QD#9:  Oct-Dec 1982
    QD#10: Jan-Mar 1983    QD#11: Apr-Jun 1983    QD#12: Jul-Sep 1983
    QD#13: Oct-Dec 1983    QD#14: Jan-Mar 1984    QD#15: Apr-Jun 1984
    QD#16: Jul-Sep 1984

AWIIe Toolkit (Don Lancaster, Synergetics)...........................$39
Quick-Trace (Anthro-Digital)............................(reg. $50)  $45
Visible Computer: 6502 (Software Masters)...............(reg. $50)  $45
ES-CAPE:  Extended S-C Applesoft Program Editor......................$60
Amper-Magic (Anthro-Digital)............................(reg. $50)  $45
Amper-Magic Volume 2 (Anthro-Digital)...................(reg. $30)  $25
"Bag of Tricks", Worth & Lechner, with diskette...........($39.95)  $36
Aztec C Compiler System (Manx Software)................(reg. $199) $180

Blank Diskettes (Verbatim)............2.25 each, or package of 20 for $40
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each.................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
                                                          or $25 per 100
   These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers............................ 6 cents each
QuikLoader EPROM System (SCRG)...............................($179) $170

Books, Books, Books..........................compare our discount prices!
    "Apple ][ Circuit Description", Gayler.................($22.95)  $21
    "Understanding the Apple II", Sather...................($22.95)  $21
    "Enhancing Your Apple II, vol. 1", Lancaster...........($15.95)  $15
                  Second edition, with //e information.
    "Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95)  $20
    "Incredible Secret Money Machine", Lancaster............($7.95)   $7
    "Beneath Apple DOS", Worth & Lechner...................($19.95)  $18
    "Beneath Apple ProDOS", Worth & Lechner................($19.95)  $18
    "What's Where in the Apple", Second Edition............($19.95)  $19
    "6502 Assembly Language Programming", Leventhal........($18.95)  $18
    "6502 Subroutines", Leventhal..........................($18.95)  $18
    "Real Time Programming -- Neglected Topics", Foster......($9.95)   $9
    "Microcomputer Graphics", Myers........................($12.95)  $12
    "Microcomputer Design & Troubleshooting", Zumchak.......($17.95)  $17

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping.  Foreign orders add postage needed.

       *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
       ***                  (214) 324-2050                  ***
       *** We accept Master Card, VISA and American Express ***
```

DP18 loads at $803 and extends well into page $25. I grabbed 30 pages, moving the Applesoft program to $2601.It thus clobbers hires screen 1 memory. If you want to use hires screen 2 and the program is too large to fit under it, use POKE 769,88 instead of POKE 769,30 in line 10. This makes the program start at $6001, and leaves $2600-3FFF totally unused.

If you want to use other ampersand routines, POKE the link address at locations 2053 and 2054 ($805 and $806). If DP18 finds an ampersand command not starting with "DP", it jumps indirectly through this vector. The vector initially contains the address of Applesoft's SYNTAX ERROR routine, but it can be changed to allow using more than one set of &-routines.

Calling DP18

Whenever you want to execute a DP18 feature, you use the "&DP" statement. If DP18 has been properly connect to the & hook at $3F5, then the & will send the computer to DP18 (at line 2430 in the listing which follows). At this point DP18 begins to analyze and execute the characters that follow the ampersand.

If the first two characters after the ampersand are not "DP", the program will jump to a vector at $805 & $806. This normally points to Applesoft's SYNTAX ERROR routine. However, this location can easily be patched to point to your own ampersand routine.

If the first two characters are correct, DP18 will analyze succeeding statements separated by colons on the same line. There must be a colon immediately after the "&DP" statement. All of the rest of the statements on the line will be executed by DP18, rather than by the normal Applesoft interpreter. If you want to shut off DP18 before the end of the line, two colons in a row with nothing between will do so.

```
150  & DP: INPUT X(0)
160  & DP:Y(0) = X(0) * X(0) * PI: PRINT Y(0) :: GOTO 150
```

It is not necessary that the "&DP:" be the first statement in a line. For example, the following statement will take the square root of a number if the two strings are equal. It uses an Applesoft string comparison, and a double precision square root.

```
170  IF A$ = "SQR" THEN  & DP:Y(0) =  SQR (X(0))
```

You can also type double precision statements as direct commands in Applesoft once DP18 has been loaded.

```
]&DP:PRINT X(0): PRINT X(0) ^ 2
```

Four types of statements can be executed by the DP18 package: assignment, INPUT, PRINT, and IF statements. INPUT and PRINT statements will be covered in a later installment.

The DP18 IF statement evaluates a logical expression in 18-digit precision, and then reverts to normal Applesoft processing:

```
180 &DP : IF A(0) < 1.52345678976543 THEN X = 3
```

The DP18 assignment statement takes two forms: real assignment, and string assignment. String assignment is used to convert DP18 values to strings, so

that they can be used by normal Applesoft:

```
190 &DP : A$ = STR$ (X(0))
```

Real assignments are the normal computational statements, like:

```
200 &DP : A(0) = (4*PI*R(0)^3)/3
```

DP18 Variables

All variables referenced by DP18 must consist of two adjacent array elements. The array must be a REAL array, that is, it must not be INTEGER or STRING.

Remember that Applesoft array subscripts begin with 0 and go up to the limit defined in the DIM statement. An array dimensioned "3,11,11" has three dimensions. The first runs from 0 to 3; the second from 0 to 11; and the third also from 0 to 11. It could contain 4*12*12=576 real elements, or 2*12*12=288 double precision elements.

Applesoft arrays are stored in memory with the leftmost subscript varying the fastest. For example, in the array XY(3,10,10), element XY(0,j,k) comes immediately before element XY(1,j,k). Therefore you may, in effect, create an array of double precision values by merely prefixing an extra dimension to the dimension list.

If you wish to set up separate variables, you may do so by dimensioning them to have two real elements each. For example, the statement

```
10  DIM A(1),B(1),C(1),X(1)
```

will set up four separate variables for use with DP18. You reference the variables within double precision statements with the subscript 0. For example:

```
20  & DP:X(0) = (A(0) + B(0)) * C(0)
```

Note that you don't have to dimension these variables, since Applesoft will default to a dimension of 10. However, it is a good idea to dimension all double precision variables because it saves memory (only 2 real elements are allocated instead of 11) and it makes it easier for someone else to follow your program.

If you wish to create an array of double precision values, you do so by dimensioning the array with one extra dimension. The extra dimension comes first and should be "1"; this dimension generates two real items, or one double precision item. For example,

```
10  DIM A(1,12),B(1,5,6)
```

creates two arrays that can be used for double precision values. The array A can be thought of as an array of 13 double precision values from A(0,0) to A(0,12). The array B could store 42 double precision values from B(0,0,0) to B(0,5,6). If you always remember to use one extra dimension, to put that extra dimension first, to set that dimension to "1", and to refer to items with the first subscript = 0, then you will succeed in using DP18.

DP18 Constants

Double precision constants are entered in the same way as single precision constants. The differences between standard Applesoft and the DP18 constants are that DP18 converts and stores 18 significant digits rather than 9, and that exponents may be in the range of +/- 63 rather than +/- 38.

Conversion of constants is very fast in DP18. DP18 will convert constants over 4 times faster than normal Applesoft, even using more digits! It is quicker to convert a constant than it is to find and use a DP18 variable, especially multi-dimensioned variables. This is completely opposite from normal Applesoft, where variables are quicker than constants.

Conversion Between Single and Double Precision

You will often need to convert a single precision value into a double precision one for purposes of computation. This is easily done by first converting it to a string and then using DP18's VAL function as shown here.

```
100  REM CONVERT X TO DOUBLE PRECISION VALUE
110  DIM DP(1)
120  INPUT "VALUE TO BE CONVERTED? ";X
130  &DP:DP(0) =  VAL ( STR$ (X))
140  &DP: PRINT DP(0)
150  GOTO 120
```

You will also want to convert from double precision back to single precision. This also involves converting to a string, but takes more than one statement.

```
100  REM CONVERT DP(0) TO SINGLE PRECISION VALUE
110  DIM DP(1)
120  &DP:INPUT "VALUE TO BE CONVERTED? ";DP(0)
130  &DP:A$ =  STR$ (DP(0))
140  X =  VAL (A$) : PRINT X
150  GOTO 120
```

Note that lines 130 and 140 could be combined onto one line if there were two colons separating the statements. See the section on functions for more information about the STR$ and VAL functions.

DP18 Arithmetic Expressions

Expressions in DP18 are very much like expressions in Applesoft. Except for AND and OR, they are evaluated using the standard rules of precedence as found on page 36 of the Applesoft manual. AND and OR have the same precedence in DP18 and are executed left to right. The order of precedence is listed below. Operations on a higher line are executed before operations on a lower line. Operators on the same line are executed left to right.

```
( ) function calls
+ - NOT          unary operators
^
* /
+ -
< >  =  <=  >=  =>  =<  <>  ><
AND OR
```

These all work the same as they do in Applesoft, except that they operate on double precision numbers.

DP18 supports many of the numerical functions that Applesoft does: SIN, COS, TAN, LOG, EXP, SGN, ABS, INT, SQR, ATN, VAL, and the string function STR$. There is also a special function, PI, which has no arguments. You don't even write parentheses after it. You just use it like it was a constant. Wherever you use it, you get the value pi accurate to 20 digits.

Explanation of the Code

As in previous installments of this series on DP18, I cannot show everything at once. A whole series of subroutines which have either already been printed or will be printed in future installments are represented in this listing by ".EQ $FFFF" in lines 1330-1550. All the data areas actually used in the code listed this month are included, so that you can see what the code is working with and on.

As mentioned above, the "&" statement sends Applesoft to line 2430. Lines 2430-2500 check for "DP" following the ampersand. If not "DP", then lines 2370-2390 branch to the next ampersand interpreter in your chain. If you have not set up another &-interpreter, then the SYNTAX ERROR message will pop out.

DP.NEXT.CMD (lines 2520-2800) begins by looking for a colon or end-of-line. End of line means you are through with DP18, so an RTS carries you back to the Applesoft interpreter. A colon means you are ready with a DP18 statement. If the next character is also a colon, however, you are sent back to Applesoft (lines 2570-2580). Next I check for the three legal tokens (IF, INPUT, and PRINT) and branch accordingly.

Since IF is simple and IF is included in this listing, let's look at IF now. Lines 3130-3280 handle the IF statement. First I evaluate the expression, which is considered to be a logical expression with a true-or-false value. Zero means false, non-zero means true. Following the expression I must find either a THEN or GOTO token. The truth value is found in DAC.EXPONENT, because a $00 exponent means a zero value. AS.IF.JUMP in the Applesoft ROMs can handle the rest, because the THEN or GOTO pops us out of DP18 back to normal Applesoft. Neat!

Meanwhile, back in DP.NEXT.CMD, if the statement is not IF-INPUT-PRINT it must be an assignment statement. If I am successful at getting a variable name next, it may be either a DP18 variable or a string variable. If AS.VALTYP is negative, it is a string variable and DP.STR takes over. If not, CHECK.DP.VAR will verify that it is a real array variable. The address is saved at RESULT, the DP18 expression evaluated, and then the answer saved at RESULT. And back to the top of DP.NEXT.CMD.

DP.STR handles statements like A$=STR$(xxx) where xxx is a DP18 expression. You can probably follow the comments in this section.

GET.A.VAR checks to see that the current character from your program is a letter, because all variables must start with a letter. If so, AS.PTRGET will search the variable tables and return with an address in the Y- and A-registers. CHECK.DP.VAR compares this address with the beginning of the array variable table. If it is inside the array table, and if the variable is real (not string or integer), it is a valid DP18 variable.

DP.EVALUATE cracks and calculates a DP18 expression. A special stack is used for temporary values, and it is deep enough to hold 10 of them. If your expression is so complicated that more than 10 temporary values need to be stacked (very unlikely), then the FORMULA TOO COMPLEX message will scream. Applesoft uses the hardware stack in page 1 for the same purpose, but it only has to stack 5-byte values; DP18 stacks 12 bytes for each value. EVALUATE starts by emptying the stack, zeroing a parenthesis level count, and clearing the accumulator (DAC). After DP.EXP finishes all the dirty work, The stack must be empty and the parenthesis level zero or there was a SYNTAX ERROR.

Actually parsing and computing an expression can be done in many ways. I chose a recursive approach that breaks the job up into little independent pieces small enough to understand. First, let's allow all expressions to be a series of relational expressions connected with ANDs and ORs. The simplest case of this is merely a relational expression alone. And the simplest relational expression is an expression all by itself with no relations. If the expression does have relational operators or ANDs or ORs, the result will be a true or false value. If not, it will have a numerical value.

Comment blocks atop DP.EXP, DP.RELAT, DP.SUM, etc. show the continued breakdown of parts of an expression. DP.RELAT connects one or more sums with relational operators. DP.SUM connects one or more terms with "+" and "-" operators. DP.TERM connects one or more factors with "*" and "/" operators. DP.FACTOR connects one or more elements with the exponentiation operator (^). DP.ELEMENT cracks a constant, searches for a variable's value, calls a function, or calls on DP.EXP recursively to handle an expression in parentheses. DP.ELEMENT also handles the unary operators "+", "-", and "NOT".

If DP.ELEMENT determines that the element is a function call, there are several types. The VAL function is supervised by lines 5800-5830. Since the argument of the VAL function is a string expression, it is significantly different from the other functions. The ATN function is also given special treatment, because DP18 allows the ATN function to be called with one or two arguments. All the rest of the functions have one DP18 expression for an argument, so they are handled as a group. A table of addresses at lines 2160-2310 directs us to the appropriate processor. The code for all these functions will be revealed in future installments.

DP.VARNUM is called upon to handle variables and numbers. First lines 6130-6280 check for and handle the special DP18 constant "PI". Lines 6300-6350 handle DP18 variables, and lines 6370-6470 handle numbers.

PUSH.DAC.STACK pushes the 12-byte value in DAC on the special expression stack, unless there is not enough room. POP.STACK.ARG pulls a 12-byte value off the stack and plops it into ARG.

And Next Month...

There are three major areas left for future installments: INPUT, PRINT, and the math functions. Some of you have been diligently studying and entering each installment as we go, and are gradually obtaining a powerful package. Others are waiting for the Quarterly Disks, to conserve their fingertips. Remember, all the source code each three months is available on disk for only $15.

```
                   *SAVE S.DP18 AMPER-LINK
                   *--------------------------------
                          .OR $803
                   *--------------------------------
                   *    APPLESOFT SUBROUTINES
                   *--------------------------------
D998-              AS.ADDON     .EQ $D998     ADD (Y) TO TXTPTR
D9DA-              AS.IF.JUMP   .EQ $D9DA     HANDLE T/F FOR IF
DD67-              AS.FRMNUM    .EQ $DD67     EVAL FP FORMULA
DEB8-              AS.CHKCLS    .EQ $DEB8     CHECK FOR )
DEBB-              AS.CHKOPN    .EQ $DEBB     CHECK FOR (
DEBE-              AS.CHKCOM    .EQ $DEBE     CHECK FOR COMMA
DEC0-              AS.SYNCHR    .EQ $DEC0     CHARACTER SCAN OR FAIL
DEC9-              AS.SYNERR    .EQ $DEC9     SYNTAX ERROR
DFE3-              AS.PTRGET    .EQ $DFE3     FIND VARIABLE
E07D-              AS.ISLETC    .EQ $E07D     LETTER CHECK
E430-              AS.FRMCPX    .EQ $E430     "FORMULA TOO COMPLEX" ERROR
E452-              AS.GETSPA    .EQ $E452     GET SPACE FOR STRING
E5E2-              AS.MOVSTR    .EQ $E5E2     MOVE STRING
                   *--------------------------------
                   *      PAGE ZERO USAGE
                   *--------------------------------
11-                AS.VALTYP    .EQ $11     LAST FAC OP 0=NUM,FF=STRING
6B-                ARYTAB       .EQ $6B,6C
71-                AS.FRESPA    .EQ $71,72
81-                VARNAM       .EQ $81,82
B1-                AS.CHRGET    .EQ $B1
B7-                AS.CHRGOT    .EQ $B7
B8-                TXTPTR       .EQ $B8,B9
F9-                P2           .EQ $F9
                   *--------------------------------
                   *   DP18 SUBROUTINES ASSEMBLED ELSEWHERE
                   *--------------------------------
FFFF-              DP.PRINT            .EQ $FFFF
FFFF-              DP.INPUT            .EQ $FFFF
FFFF-              FIN                 .EQ $FFFF
FFFF-              DP.SGN              .EQ $FFFF
FFFF-              DP.INT              .EQ $FFFF
FFFF-              DP.ABS              .EQ $FFFF
FFFF-              DP.SQR              .EQ $FFFF
FFFF-              DP.LOGE             .EQ $FFFF
FFFF-              DP.EXPE             .EQ $FFFF
FFFF-              DP.COS              .EQ $FFFF
FFFF-              DP.SIN              .EQ $FFFF
FFFF-              DP.TAN              .EQ $FFFF
FFFF-              MOVE.DAC.YA         .EQ $FFFF
FFFF-              QUICK.FOUT          .EQ $FFFF
FFFF-              DP.POWER            .EQ $FFFF
FFFF-              DSUB                .EQ $FFFF
FFFF-              DADD                .EQ $FFFF
FFFF-              DMULT               .EQ $FFFF
FFFF-              DDIV                .EQ $FFFF
FFFF-              DP.ATN              .EQ $FFFF
FFFF-              DP.VAL              .EQ $FFFF
FFFF-              MOVE.YA.DAC         .EQ $FFFF
FFFF-              MOVE.YA.DAC.1       .EQ $FFFF
                   *--------------------------------
                   *   AMPERSAND VECTORS
                   *--------------------------------
0803- EE 08                .DA DP18      STARTING ADDRESS FOR &-INTERPRETER
0805- C9 DE        AMP.LINK .DA AS.SYNERR    LINK TO NEXT &-INTERPRETER
                   *--------------------------------
                   *      WORK AREAS FOR DPFP
                   *--------------------------------
0807-              WORK             .EQ *
0807-              SGNEXP           .BS 1
0808-              EXP              .BS 1
0809-              DGTCNT           .BS 1
080A-              DECFLG           .BS 1
                   *--------------------------------
080B-              DAC              .BS 12
080B-              DAC.EXPONENT     .EQ DAC
080C-              DAC.HI           .EQ DAC+1
0815-              DAC.EXTENSION    .EQ DAC+10
0816-              DAC.SIGN         .EQ DAC+11
                   *--------------------------------
10-                WRKSZ            .EQ *-WORK
                   *--------------------------------
0817-              ARG              .BS 12
                   *--------------------------------
                   *BUFFER FOR 'FOUT' AND
                   *LARGE ACC FOR MULTIPLICATION
                   *--------------------------------
0823-              FOUT.BUF         .BS 41
29-                FOUT.BUF.SIZE    .EQ *-FOUT.BUF
0823-              MAC              .EQ FOUT.BUF
                   *--------------------------------
78-                STACK.SIZE       .EQ 12*10 10 ENTRIES BEFORE
084C-              STACK.PNTR       .BS 1                OVERFLOW
084D-              STACK            .BS STACK.SIZE
08C5-              RPAREN.CNT       .BS 1
                   *--------------------------------
08C6-              REL.OPS          .BS 1
08C7-              RESULT           .BS 2
08C9-              INDEX            .BS 1
                   *--------------------------------
                   *       TOKEN ASSIGNMENTS
                   *--------------------------------
C8-                TKN.PLUS     .EQ 200   +
C9-                TKN.MINUS    .EQ 201   -
CA-                TKN.STAR     .EQ 202   *
CB-                TKN.SLASH    .EQ 203   /
CC-                TKN.POWER    .EQ 204   ^
D0-                TKN.EQUAL    .EQ 208   =
BA-                TKN.PRINT    .EQ 186   PRINT
84-                TKN.INPUT    .EQ 132   INPUT
E4-                TKN.STR      .EQ 228   STR$
AD-                TKN.IF       .EQ 173   IF
C4-                TKN.THEN     .EQ 196   THEN
AB-                TKN.GOTO     .EQ 171   GOTO
C6-                TKN.NOT      .EQ 198   NOT
CD-                TKN.AND      .EQ 205   AND
CE-                TKN.OR       .EQ 206   OR
                   *--------------------------------
                   *       JMP TABLE FOR FUNCTIONS
                   *--------------------------------
                   DP.FUNC
08CA- FE FF               .DA DP.SGN-1         SGN (TKN 210)
08CC- FE FF               .DA DP.INT-1         INT
08CE- FE FF               .DA DP.ABS-1         ABS
08D0- C8 DE               .DA AS.SYNERR-1      USR
08D2- C8 DE               .DA AS.SYNERR-1      FRE
08D4- C8 DE               .DA AS.SYNERR-1      SCRN(
08D6- C8 DE               .DA AS.SYNERR-1      PDL
08D8- C8 DE               .DA AS.SYNERR-1      POS
08DA- FE FF               .DA DP.SQR-1         SQR
08DC- C8 DE               .DA AS.SYNERR-1      RND
08DE- FE FF               .DA DP.LOGE-1        LOG   #220
08E0- FE FF               .DA DP.EXPE-1        EXP
```

```
08E2- FE FF          .DA DP.COS-1         COS
08E4- FE FF          .DA DP.SIN-1         SIN
08E6- FE FF          .DA DP.TAN-1         TAN
               *     ATN HANDLED SPECIALLY
               *--------------------------------
               *--------------------------------
               *     &-INTERPRETER FOR DP18
               *--------------------------------
               NOT.DP18.CALL
08E8- 20 B7 00       JSR AS.CHRGOT
08EB- 6C 05 08       JMP (AMP.LINK)  SYNTAX ERROR OR NEXT CHAINED
                                                       &-ROUTINE
               *--------------------------------
               *   & ENTRY POINT
               *--------------------------------
08EE- C9 44    DP18  CMP #'D'      CHECK FOR "DP:" AFTER "&"
08F0- D0 F6          BNE NOT.DP18.CALL
08F2- A0 01          LDY #1
08F4- B1 B8          LDA (TXTPTR),Y
08F6- C9 50          CMP #'P'
08F8- D0 EE          BNE NOT.DP18.CALL
08FA- C8             INY           ADD 2 TO TXTPTR, TO POINT
08FB- 20 98 D9       JSR AS.ADDON      AT NEXT CHAR AFTER "&DP"
               *--------------------------------
               DP.NEXT.CMD
08FE- 20 B7 00       JSR AS.CHRGOT  SEE IF EOL
0901- D0 3F          BNE DP.SYNERR.1  ...NEITHER COLON NOR EOL
0903- A8             TAY           CHECK FOR EOL
0904- F0 3B          BEQ .3        ...EOL, SO RETURN
0906- 20 B1 00       JSR AS.CHRGET     CHARACTER AFTER COLON
0909- F0 36          BEQ .3        ...COLON OR EOL
090B- C9 BA          CMP #TKN.PRINT
090D- F0 2C          BEQ .1
090F- C9 84          CMP #TKN.INPUT
0911- F0 2B          BEQ .2
0913- C9 AD          CMP #TKN.IF
0915- F0 6C          BEQ DP.IF
0917- 20 9B 09       JSR GET.A.VAR GET ADDRESS OF VAR
091A- A6 11          LDX AS.VALTYP
091C- 30 27          BMI DP.STR  STRING VAR
091E- 20 A3 09       JSR CHECK.DP.VAR
0921- 8C C8 08       STY RESULT+1   SAVE ADRS OF VARIABLE
0924- 8D C7 08       STA RESULT
0927- A9 D0          LDA #TKN.EQUAL     NEXT CHAR MUST BE "="
0929- 20 C0 DE       JSR AS.SYNCHR   OR ELSE SYNTAX ERROR
092C- 20 B2 09       JSR DP.EVALUATE
092F- AD C7 08       LDA RESULT
0932- AC C8 08       LDY RESULT+1
0935- 20 FF FF       JSR MOVE.DAC.YA
0938- 4C FE 08       JMP DP.NEXT.CMD
093B- 4C FF FF .1    JMP DP.PRINT
093E- 4C FF FF .2    JMP DP.INPUT
0941- 60       .3    RTS
               *--------------------------------
               DP.SYNERR.1
0942- 4C C9 DE       JMP AS.SYNERR
               *--------------------------------
               *   <STRING> = STR$(<DPEXP>)
               *--------------------------------
0945- 85 F9    DP.STR STA P2              SAVE ADDR OF STRING
0947- 84 FA          STY P2+1                        VARIABLE
0949- A9 D0          LDA #TKN.EQUAL      MUST HAVE "="
094B- 20 C0 DE       JSR AS.SYNCHR
094E- A9 E4          LDA #TKN.STR        MUST HAVE "STR$"
0950- 20 C0 DE       JSR AS.SYNCHR
0953- 20 BB DE       JSR AS.CHKOPN       MUST HAVE "("
0956- 20 B2 09       JSR DP.EVALUATE     GET EXPRESSION
0959- 20 B8 DE       JSR AS.CHKCLS       MUST HAVE ")"
095C- 20 FF FF       JSR QUICK.FOUT      CONVERT TO SIMPLE STR$ FORMAT
095F- CE C9 08       DEC INDEX           DON'T COUNT TRAILING $00 BYTE
0962- AD C9 08       LDA INDEX           GET LENGTH
0965- 20 52 E4       JSR AS.GETSPA       GET SPACE IN STRING AREA
0968- A0 00          LDY #0              MOVE DATA INTO VARIABLE
096A- 91 F9          STA (P2),Y          LENGTH
096C- A5 71          LDA AS.FRESPA
096E- C8             INY
096F- 91 F9          STA (P2),Y          LO ADDRESS
0971- A5 72          LDA AS.FRESPA+1
0973- C8             INY
0974- 91 F9          STA (P2),Y          HI ADDRESS
0976- A2 23          LDX #FOUT.BUF       COPY STRING DATA INTO PLACE
0978- A0 08          LDY /FOUT.BUF
097A- AD C9 08       LDA INDEX
097D- 20 E2 E5       JSR AS.MOVSTR
0980- 4C FE 08       JMP DP.NEXT.CMD
               *--------------------------------
               *   IF <DPEXP> THEN <NORMAL STATEMENTS>
               *   IF <DPEXP> THEN <LINE #>
               *   IF <DPEXP> GOTO <LINE #>
               *--------------------------------
0983- 20 B1 00 DP.IF JSR AS.CHRGET      GOBBLE THE IF
0986- 20 B2 09       JSR DP.EVALUATE     GET THE EXPRESSION
0989- 20 B7 00       JSR AS.CHRGOT       GET NEXT CHAR
098C- C9 AB          CMP #TKN.GOTO       GOTO?
098E- F0 05          BEQ .1              ...YES
0990- A9 C4          LDA #TKN.THEN       ...NO, TRY "THEN"
0992- 20 C0 DE       JSR AS.SYNCHR
0995- AD 0B 08 .1    LDA DAC.EXPONENT    GET RESULT OF EXPRESSION
0998- 4C DA D9       JMP AS.IF.JUMP      LET APPLESFOT FIRMWARE DO IT
               * AS.IF.JUMP COMPARES ACC TO 0. IF 0, IT SKIPS
               * TO NEXT PROG. LINE. IF # 0, IT EXECUTES NEXT STATEMENT
               *--------------------------------
               *   GET VARIABLE NAME AND ADDRESS
               *--------------------------------
               GET.A.VAR
099B- 20 7D E0       JSR AS.ISLETC      1ST CHAR MUST BE LETTER
099E- 90 A2          BCC DP.SYNERR.1        NO, SYN ERR
09A0- 4C E3 DF       JMP AS.PTRGET GET ADRS OF VAR
               *--------------------------------
               *   CHECK IF VALID DP18 VARIABLE
               *      ASSUME THIS ROUTINE CALLED AFTER "GET.A.VAR"
               *      A DP18 VARIABLE MUST BE A REAL ARRAY
               *--------------------------------
               CHECK.DP.VAR
09A3- C4 6C          CPY ARYTAB+1        BE SURE IT IS AN ARRAY
09A5- 90 9B          BCC DP.SYNERR.1        NO, SYNTAX ERROR
09A7- D0 04          BNE .1          YES, AN ARRAY
09A9- C5 6B          CMP ARYTAB
09AB- 90 95          BCC DP.SYNERR.1        NOT AN ARRAY, SYNTAX ERROR
09AD- 24 82    .1    BIT VARNAM+1 BE SURE FLOATING POINT
09AF- 30 91          BMI DP.SYNERR.1        NO, SYNTAX ERROR
09B1- 60             RTS
               *--------------------------------
               *   EVALUATE DP18 EXPRESSION
               *--------------------------------
               DP.EVALUATE
09B2- A9 00          LDA #0              START WITH EMPTY STACK
09B4- 8D C5 08       STA RPAREN.CNT      ...AND NO PARENTHESES
09B7- 8D 4C 08       STA STACK.PNTR
09BA- 20 64 0B       JSR DP.ZERO         ZERO TO DAC
09BD- 20 C9 09       JSR DP.EXP          EVALUATE AN EXPRESSION
09C0- AD 4C 08       LDA STACK.PNTR      SHOULD BE BACK TO EMPTY STACK
09C3- 0D C5 08       ORA RPAREN.CNT      AND NO PARENTHESES
```

```
09C6- D0 37     4610        BNE DP.SYNERR.2   ...SYNTAX ERROR
09C8- 60        4620        RTS               ...ALL OKAY!
                4630 *--------------------------------
                4640 *   GENERAL EXPRESSION
                4650 *      EXP = RELAT
                4660 *      EXP = EXP LOGOP RELAT
                4670 *    LOGOP = "AND" OR "OR"
                4680 *--------------------------------
09C9- 20 02 0A  4690 DP.EXP JSR DP.RELAT
09CC- 20 B7 00  4700 .1     JSR AS.CHRGOT
09CF- C9 CD     4710        CMP #TKN.AND
09D1- F0 12     4720        BEQ .3
09D3- C9 CE     4730        CMP #TKN.OR
09D5- D0 27     4740        BNE .6          ...FINISHED
                4750 *---<EXP> OR <EXP>---------------
09D7- 20 F3 09  4760        JSR .5          GET NEXT RELAT
09DA- 0D 0B 08  4770        ORA DAC.EXPONENT
09DD- D0 0E     4780        BNE .4          ...TRUE
09DF- 20 64 0B  4790 .2     JSR DP.FALSE ...FALSE
09E2- 4C CC 09  4800        JMP .1
                4810 *---<EXP> AND <EXP>--------------
09E5- 20 F3 09  4820 .3     JSR .5          GET NEXT RELAT
09E8- 2D 0B 08  4830        AND DAC.EXPONENT
09EB- F0 F2     4840        BEQ .2          ...FALSE
09ED- 20 6F 0B  4850 .4     JSR DP.TRUE   ...TRUE
09F0- 4C CC 09  4860        JMP .1
                4870 *---GET <EXP> AFTER RELOP--------
09F3- AD 0B 08  4880 .5     LDA DAC.EXPONENT
09F6- 48        4890        PHA
09F7- 20 B1 00  4900        JSR AS.CHRGET
09FA- 20 02 0A  4910        JSR DP.RELAT
09FD- 68        4920        PLA
09FE- 60        4930 .6     RTS
                4940 *--------------------------------
                4950 DP.SYNERR.2
09FF- 4C C9 DE  4960        JMP AS.SYNERR
                4970 *--------------------------------
                4980 *   RELATIONAL EXPRESSION
                4990 *      RELAT = SUM
                4000 *      RELAT = RELAT RELOP SUM
                4010 *      RELOP = "<", "=", ">", "<=", "=<", ">=",
                4020 *              "=>", "<>", OR "><"
                4030 *--------------------------------
                4040 DP.RELAT
0A02- 20 6F 0A  4050        JSR DP.SUM    GET <EXP>
0A05- A9 00     4060 .1     LDA #0
0A07- 8D C6 08  4070        STA REL.OPS
0A0A- 20 B7 00  4080        JSR AS.CHRGOT
0A0D- 38        4090 .2     SEC          > IS $CF, = IS $D0, < IS $D1
0A0E- E9 CF     4100        SBC #$CF     > IS 0, = IS 1, < IS 2
0A10- 90 1A     4110        BCC .4       ...NOT RELOP
0A12- C9 03     4120        CMP #$03
0A14- B0 16     4130        BCS .4       ...NOT RELOP
0A16- 2A        4140        ROL          > IS 0, = IS 2, < IS 4
0A17- D0 02     4150        BNE .3       4 OR 2
0A19- A9 01     4160        LDA #1       > IS 1
0A1B- 4D C6 08  4170 .3     EOR REL.OPS  SET BITS IN REL.OPS: 00000<=>
0A1E- CD C6 08  4180        CMP REL.OPS  CHECK FOR REPEATED OPS
0A21- 90 DC     4190        BCC DP.SYNERR.2   ...YES, SYNTAX ERROR
0A23- 8D C6 08  4200        STA REL.OPS
0A26- 20 B1 00  4210        JSR AS.CHRGET     GET NEXT CHAR
0A29- 4C 0D 0A  4220        JMP .2       CHECK FOR <=> AGAIN
                4230 *---PERFORM RELOP----------------
0A2C- AD C6 08  4240 .4     LDA REL.OPS  WERE THERE ANY?
0A2F- F0 3D     4250        BEQ .8       NO, RETURN
0A31- C9 07     4260        CMP #7       ALL THREE OPS?
0A33- F0 CA     4270        BEQ DP.SYNERR.2  ...YES, SYNTAX ERROR
0A35- 20 C1 0B  4280        JSR PUSH.DAC.STACK  SAVE EXP1
0A38- 20 6F 0A  4290        JSR DP.SUM          GET NEXT EXP2
0A3B- 20 DD 0B  4300        JSR POP.STACK.ARG   GET EXP1 IN ARG
0A3E- 20 FF FF  4310        JSR DSUB            FORM EXP1 - EXP2
0A41- AD 0B 08  4320        LDA DAC.EXPONENT
0A44- F0 0E     4330        BEQ .45             EXP1 = EXP2
0A46- AD 16 08  4340        LDA DAC.SIGN
0A49- 30 16     4350        BMI .6              EXP1 < EXP2
0A4B- AD C6 08  4360        LDA REL.OPS         EXP1 > EXP2
0A4E- 29 01     4370        AND #$01            ">" OPERATOR?
0A50- F0 16     4380        BEQ .7              ...NO, FALSE
0A52- D0 07     4390        BNE .5              ...YES, TRUE
0A54- AD C6 08  4400 .45    LDA REL.OPS  EXP1 = EXP2
0A57- 29 02     4410        AND #$02     "=" OPERATOR?
0A59- F0 0D     4420        BEQ .7       ...NO, FALSE
0A5B- 20 6F 0B  4430 .5     JSR DP.TRUE  ...YES, TRUE
0A5E- 4C 05 0A  4440        JMP .1
0A61- AD C6 08  4450 .6     LDA REL.OPS  EXP1 < EXP2
0A64- 29 02     4460        AND #$02     "<" OPERATOR?
0A66- D0 F3     4470        BNE .5       ...YES, TRUE
0A68- 20 64 0B  4480 .7     JSR DP.FALSE ...NO, FALSE
0A6B- 4C 05 0A  4490        JMP .1
0A6E- 60        4500 .8     RTS
                4510 *--------------------------------
                4520 *   SUMMATION
                4530 *      SUM = TERM
                4540 *      SUM = SUM ADDOP TERM
                4550 *    ADDOP = "+" OR "-"
                4560 *--------------------------------
0A6F- 20 A5 0A  4570 DP.SUM JSR DP.TERM
0A72- 20 B7 00  4580 .1     JSR AS.CHRGOT
0A75- C9 C8     4590        CMP #TKN.PLUS
0A77- F0 0D     4600        BEQ .3        +
0A79- C9 2B     4610        CMP #'+
0A7B- F0 09     4620        BEQ .3        +
0A7D- C9 C9     4630        CMP #TKN.MINUS
0A7F- F0 06     4640        BEQ .4        -
0A81- C9 2D     4650        CMP #'-
0A83- F0 02     4660        BEQ .4        -
0A85- 60        4670        RTS           END OF EXP
0A86- 18        4680 .3     CLC           .CC. FOR +, .CS. FOR -
0A87- 08        4690 .4     PHP           SAVE WHETHER + OR -
0A88- 20 C1 0B  4700        JSR PUSH.DAC.STACK
0A8B- 20 B1 00  4710        JSR AS.CHRGET
0A8E- 20 A5 0A  4720        JSR DP.TERM
0A91- 20 DD 0B  4730        JSR POP.STACK.ARG
0A94- 28        4740        PLP           .CC. FOR +, .CS. FOR -
0A95- 90 08     4750        BCC .5
0A97- AD 16 08  4760        LDA DAC.SIGN
0A9A- 49 FF     4770        EOR #$FF
0A9C- 8D 16 08  4780        STA DAC.SIGN
0A9F- 20 FF FF  4790 .5     JSR DADD
0AA2- 4C 72 0A  4800        JMP .1
                4810 *--------------------------------
                4820 *   TERMS OF A SUMMATION
                4830 *      TERM = FACTOR
                4840 *      TERM = TERM MULOP FACTOR
                4850 *    MULOP = "*" OR "/"
                4860 *--------------------------------
                4870 DP.TERM
0AA5- 20 D1 0A  4880        JSR DP.FACTOR
0AA8- 20 B7 00  4890 .1     JSR AS.CHRGOT
0AAB- C9 CA     4900        CMP #TKN.STAR     *?
```

```
0AAD- F0 05    4910        BEQ .2
0AAF- C9 CB    4920        CMP #TKN.SLASH    / ?
0AB1- F0 02    4930        BEQ .3
0AB3- 60       4940        RTS
0AB4- 18       4950 .2     CLC             .CC. FOR *, .CS. FOR /
0AB5- 08       4960 .3     PHP             SAVE * OR / FLAG
0AB6- 20 C1 0B 4970        JSR PUSH.DAC.STACK
0AB9- 20 B1 00 4980        JSR AS.CHRGET
0ABC- 20 D1 0A 4990        JSR DP.FACTOR
0ABF- 20 DD 0B 5000        JSR POP.STACK.ARG
0AC2- 28       5010        PLP             GET * OR / FLAG
0AC3- B0 06    5020        BCS .4          .../
0AC5- 20 FF FF 5030        JSR DMULT       ...*
0AC8- 4C A8 0A 5040        JMP .1
0ACB- 20 FF FF 5050 .4     JSR DDIV
0ACE- 4C A8 0A 5060        JMP .1
               5070 *--------------------------------
               5080 *   FACTORS OF A TERM
               5090 *      FACTOR = ELEMENT
               5100 *      FACTOR = FACTOR ^ ELEMENT
               5110 *--------------------------------
               5120 DP.FACTOR
0AD1- 20 B7 00 5130        JSR AS.CHRGOT
0AD4- 20 F1 0A 5140        JSR DP.ELEMENT.1
0AD7- 20 B7 00 5150 .1     JSR AS.CHRGOT
0ADA- C9 CC    5160        CMP #TKN.POWER   ^?
0ADC- F0 01    5170        BEQ .2
0ADE- 60       5180        RTS              NO
0ADF- 20 C1 0B 5190 .2     JSR PUSH.DAC.STACK
0AE2- 20 EE 0A 5200        JSR DP.ELEMENT
0AE5- 20 DD 0B 5210        JSR POP.STACK.ARG
0AE8- 20 FF FF 5220        JSR DP.POWER
0AEB- 4C D7 0A 5230        JMP .1
               5240 *--------------------------------
               5250 *   ELEMENTS OF A FACTOR
               5260 *      ELEMENT = NUMBER, VARIABLE, OR FUNCTION()
               5270 *      ELEMENT = (EXP)
               5280 *      ELEMENT = UNARY ELEMENT
               5290 *         UNARY = "+" OR "-" OR "NOT"
               5300 *--------------------------------
               5310 DP.ELEMENT
0AEE- 20 B1 00 5320        JSR AS.CHRGET
               5330 DP.ELEMENT.1
0AF1- C9 C8    5340        CMP #TKN.PLUS     CHECK FOR UNARY +
0AF3- F0 F9    5350        BEQ DP.ELEMENT    ...YES, JUST IGNORE IT
0AF5- C9 C9    5360        CMP #TKN.MINUS    CHECK FOR UNARY -
0AF7- D0 0C    5370        BNE .1            ...NO
0AF9- 20 EE 0A 5380        JSR DP.ELEMENT    GET THE EXP VALUE (RECURSIVE
0AFC- AD 16 08 5390        LDA DAC.SIGN   AND NEGATE IT               CALL)
0AFF- 49 FF    5400        EOR #$FF
0B01- 8D 16 08 5410        STA DAC.SIGN
0B04- 60       5420        RTS
               5430 *---CHECK FOR (EXP)--------------
0B05- C9 28    5440 .1     CMP #'(
0B07- D0 10    5450        BNE .2          ...NO
0B09- EE C5 08 5460        INC RPAREN.CNT
0B0C- 20 B1 00 5470        JSR AS.CHRGET  GET 1ST CHAR OF EXP
0B0F- 20 C9 09 5480        JSR DP.EXP      (EXP)
0B12- 20 B8 DE 5490        JSR AS.CHKCLS
0B15- CE C5 08 5500        DEC RPAREN.CNT
0B18- 60       5510        RTS
               5520 *---TRY VARIOUS FUNCTIONS--------
0B19- A8       5530 .2     TAY               SEE IF FUNCTION
0B1A- 10 5D    5540        BPL DP.VARNUM     ...NO, TRY NUMBER OR VARIABLE
0B1C- C9 C6    5550        CMP #TKN.NOT      "NOT"?
0B1E- F0 3C    5560        BEQ .5            ...YES
0B20- C9 D2    5570        CMP #210          CHECK RANGE
0B22- 90 52    5580        BCC DP.SYNERR.3   ...NOT VALID DP FUNCTION
0B24- C9 E5    5590        CMP #229          MAY BE "VAL"
0B26- F0 28    5600        BEQ .4            ...VAL(STRING)
0B28- C9 E1    5610        CMP #225          ATN?              OTHERS
0B2A- 90 05    5620        BCC .3            ...NO, BUT IN RANGE FOR
0B2C- D0 48    5630        BNE DP.SYNERR.3   ...NOT VALID DP18 FUNCTION
0B2E- 4C FF FF 5640        JMP DP.ATN
0B31- E9 D1    5650 .3     SBC #209          CARRY CLEAR SUBS 1 MORE
0B33- 0A       5660        ASL               MULT BY 2
0B34- A8       5670        TAY               INDEX INTO TABLE
0B35- B9 CB 08 5680        LDA DP.FUNC+1,Y   GET HI ADR
0B38- 48       5690        PHA
0B39- B9 CA 08 5700        LDA DP.FUNC,Y     GET LO ADR
0B3C- 48       5710        PHA
0B3D- 20 B1 00 5720        JSR AS.CHRGET
0B40- 20 BB DE 5730        JSR AS.CHKOPN MUST HAVE (
0B43- EE C5 08 5740        INC RPAREN.CNT
0B46- 20 C9 09 5750        JSR DP.EXP    EVALUATE ARG
0B49- 20 B8 DE 5760        JSR AS.CHKCLS
0B4C- CE C5 08 5770        DEC RPAREN.CNT
0B4F- 60       5780        RTS           EVALUATES FUNCTION
               5790 *---"VAL" FUNCTION---------------
0B50- 20 B1 00 5800 .4     JSR AS.CHRGET
0B53- 20 BB DE 5810        JSR AS.CHKOPN
0B56- 20 FF FF 5820        JSR DP.VAL
0B59- 4C B8 DE 5830        JMP AS.CHKCLS
               5840 *---"NOT" ELEMENT----------------
0B5C- 20 EE 0A 5850 .5     JSR DP.ELEMENT    GET ARGUMENT (RECURSIVE
0B5F- AD 0B 08 5860        LDA DAC.EXPONENT                      CALL)
0B62- F0 0B    5870        BEQ DP.TRUE
               5880 *      FALL INTO DP.FALSE
               5890 *--------------------------------
               5900 DP.ZERO
               5910 DP.FALSE
0B64- A9 00    5920        LDA #0        FALSE, PUT 0 IN DAC
0B66- A0 0B    5930        LDY #11
0B68- 99 0B 06 5940 .1     STA DAC,Y
0B6B- 88       5950        DEY
0B6C- 10 FA    5960        BPL .1
0B6E- 60       5970        RTS
               5980 *--------------------------------
               5990 DP.TRUE
0B6F- A9 F2    6000        LDA #CON.ONE      TRUE, PUT 1 IN DAC
0B71- A0 0B    6010        LDY /CON.ONE
0B73- 4C FF FF 6020        JMP MOVE.YA.DAC
               6030 *--------------------------------
0B76- 4C C9 DE 6040 DP.SYNERR.3 JMP AS.SYNERR
               6050 *--------------------------------
               6060 *   VARIABLE OR NUMBER
               6070 *      VARNUM = DP18 VARIABLE
               6080 *      VARNUM = NUMBER
               6090 *      VARNUM = NEGOP NUMBER
               6100 *      VARNUM = "PI"
               6110 *--------------------------------
               6120 DP.VARNUM
0B79- A0 00    6130        LDY #0
0B7B- B1 B8    6140        LDA (TXTPTR),Y
0B7D- C9 50    6150        CMP #'P       CHECK FOR PI
0B7F- D0 18    6160        BNE .1
0B81- C8       6170        INY           Y=1
0B82- B1 B8    6180        LDA (TXTPTR),Y
0B84- C9 49    6190        CMP #'I
0B86- D0 11    6200        BNE .1
0B88- C8       6210        INY           Y=2
```

```
0B89- B1 B8     6220        LDA (TXTPTR),Y
0B8B- C9 28     6230        CMP #'(       MUST NOT BE ARRAY
0B8D- F0 0A     6240        BEQ .1
0B8F- 20 98 D9  6250        JSR AS.ADDON       ADVANCE TXTPTR PAST "PI"
0B92- A9 FD     6260        LDA #CON.PI
0B94- A0 0B     6270        LDY /CON.PI
0B96- 4C FF FF  6280        JMP MOVE.YA.DAC.1 GET PI INTO DAC W/GUARD DIGITS
                6290 *---CHECK FOR VARIABLE-----------
0B99- 20 B7 00  6300 .1     JSR AS.CHRGOT
0B9C- 20 7D E0  6310        JSR AS.ISLETC
0B9F- 90 09     6320        BCC .2             ...NOT LETTER, TRY NUMBER
0BA1- 20 E3 DF  6330        JSR AS.PTRGET      ...LETTER, GET VARIABLE ADDR
0BA4- 20 A3 09  6340        JSR CHECK.DP.VAR   BE SURE IT IS REAL ARRAY
0BA7- 4C FF FF  6350        JMP MOVE.YA.DAC    GET VALUE INTO DAC
                6360 *---CHECK FOR NUMBER-------------
0BAA- C9 2E     6370 .2     CMP #'.       DECIMAL POINT?
0BAC- F0 10     6380        BEQ .3        YES
0BAE- C9 C8     6390        CMP #TKN.PLUS PLUS?
0BB0- F0 0C     6400        BEQ .3        YES
0BB2- C9 C9     6410        CMP #TKN.MINUS MINUS?
0BB4- F0 08     6420        BEQ .3        YES
0BB6- C9 30     6430        CMP #'0
0BB8- 90 BC     6440        BCC DP.SYNERR.3       NOT A DIGIT
0BBA- C9 3A     6450        CMP #'9+1
0BBC- B0 B8     6460        BCS DP.SYNERR.3       NOT A DIGIT
0BBE- 4C FF FF  6470 .3     JMP FIN       CONVERT NUMBER
                6480 *--------------------------------
                6490 *      PUSH (DAC) ONTO EXPRESSION STACK
                6500 *--------------------------------
                6510 PUSH.DAC.STACK
0BC1- AC 4C 08  6520        LDY STACK.PNTR
0BC4- C0 6C     6530        CPY #STACK.SIZE-12
0BC6- B0 12     6540        BCS .2        STACK ALREADY FULL
0BC8- A2 00     6550        LDX #0
0BCA- BD 0B 08  6560 .1     LDA DAC,X
0BCD- 99 4D 08  6570        STA STACK,Y
0BD0- C8        6580        INY
0BD1- E8        6590        INX
0BD2- E0 0C     6600        CPX #12       STACK 12 BYTES
0BD4- 90 F4     6610        BCC .1
0BD6- 8C 4C 08  6620        STY STACK.PNTR
0BD9- 60        6630        RTS
0BDA- 4C 30 E4  6640 .2     JMP AS.FRMCPX  FORMULA TOO COMPLEX
                6650 *--------------------------------
                6660 *      POP EXPRESSION STACK INTO ARG
                6670 *--------------------------------
                6680 POP.STACK.ARG
0BDD- AC 4C 08  6690        LDY STACK.PNTR
0BE0- F0 94     6700        BEQ DP.SYNERR.3    STACK IS EMPTY
0BE2- A2 0B     6710        LDX #11
0BE4- 88        6720 .1     DEY
0BE5- B9 4D 08  6730        LDA STACK,Y
0BE8- 9D 17 08  6740        STA ARG,X
0BEB- CA        6750        DEX
0BEC- 10 F6     6760        BPL .1
0BEE- 8C 4C 08  6770        STY STACK.PNTR
0BF1- 60        6780        RTS
                6790 *--------------------------------
0BF2- 41 10 00
0BF5- 00 00 00
0BF8- 00 00 00
0BFB- 00 00     6800 CON.ONE    .HS 41100000000000000000
0BFD- 41 31 41
0C00- 59 26 53
0C03- 58 97 93
0C06- 23 85     6810 CON.PI     .HS 4131415926535897932385
```

Faster Amper-routine to Zero Arrays.........Johan Zwiekhorst
Maasmechelen, Belgium

Although I have never subscribed to Apple Assembly Line, a friend of mine (who lives in nearby Heerlen, the Netherlands) does, and I always read his copies.

A few days ago I needed a routine to clear to zero all the elements in a number of Applesoft arrays, so I started looking in my friend's collection of AAL for such a program. I found the article entitled "Save Garbage by Emptying Arrays" in the December 1982 issue, pages 22-25.

That routine, however, only cleared string arrays. Bob designed it to set all strings in an array to null strings, so that garbage collection would be faster. But I needed a fast way to clear integer and real arrays as well. Bob's routine was also limited to clearing one array per call.

My routine clears any type of arrays, and can accept a list of array names separated by commas. It uses the ampersand hook, like this:

```
        & CLEAR array1,array2,array3,...
```

You can load the routine in any available memory, anywhere you have a spare 79 bytes. The listing shows it assembled into the ever-popular $300 space, but there are no internal addresses which require it to be there. Just be sure you hook the ampersand to the program, wherever you put it. If it is at $300, hook it like this:

```
        POKE 1013,76 : POKE 1014,0 : POKE 1015,3
```

The program is very similar to Bob's 1982 version: I eliminated the check he made for string arrays, added ampersand control, and checked for a comma to allow a list of array names rather than just one.

Lines 1250-1260 check that the byte following the ampersand is the CLEAR token. If not, a SYNTAX ERROR will result. If it is CLEAR, all is well.

Lines 1280-1290 check for a comma, and are not used until we have finished clearing an array. At the end, lines 1690-1710, you find my test after clearing an array. If the next byte of program is not a colon or end of line, it will branch back to the comma-test.

The code in between zeroes all the data bytes in an array. I could have done it the same way Bob did, but I did change a few things. Compare mine with his and you will learn two ways to control a clearing loop.

How about a complete example of using &CLEAR? Lets make three arrays, with a mixture of types and dimensions. Of course, when the DIM statement works it initially zeroes the arrays, but I needed them cleared again later on.

```
100 DIM A(10,20), B%(200,4,4), C$(20)
110 PRINT CHR$(4)"BLOAD B.CLEAR ARRAYS,A$300"
120 POKE 1013,76:POKE1014,0:POKE1015,3
...
500 &CLEAR A,B%,C$
```

```
                    1000 *SAVE S.CLEAR ARRAYS
                    1010 *--------------------------------
                    1020 *    &CLEAR <ARRAY LIST>
                    1030 *       SETS ALL VALUES IN REAL ARRAYS TO 0
                    1040 *                    INTEGER ARRAYS TO 0
                    1050 *                     STRING ARRAYS TO ""
                    1060 *--------------------------------
                    1070 *    WRITTEN BY JOHAN ZWIEKHORST, BASED ON
                    1080 *    "CLEAR STRING ARRAY" BY BOB SANDER-CEDERLOF
                    1090 *    IN DECEMBER, 1982 APPLE ASSEMBLY LINE
                    1100 *--------------------------------
BD-                 1110 T.CLEAR    .EQ $BD   "CLEAR" TOKEN
                    1120 *--------------------------------
94-                 1130 ARYPT      .EQ $94
9B-                 1140 LOWTR      .EQ $9B
9D-                 1150 ARYEND     .EQ $9D   (= FAC)
B1-                 1170 CHRGET     .EQ $B1
B7-                 1180 CHRGOT     .EQ $B7
DEC9-               1190 SYNERR     .EQ $DEC9
F7D9-               1200 GETARYPT   .EQ $F7D9
                    1210 *--------------------------------
                    1220        .OR $300      (COULD BE ANYWHERE YOU LIKE)
                    1240 CLEAR.ARRAYS
0300- C9 BD         1250        CMP #T.CLEAR       &CLEAR?
0302- F0 07         1260        BEQ .3             ...YES
0304- 4C C9 DE      1270 .1     JMP SYNERR
0307- C9 2C         1280 .2     CMP #$2C           COMMA?
0309- D0 F9         1290        BNE .1
                    1300 *---GET STARTING ADDRESS---------
030B- 20 B1 00      1310 .3     JSR CHRGET    GET NEXT CHAR (SHOULD BE LETTER)
030E- 20 D9 F7      1320        JSR GETARYPT  FIND NAME/ADDRESS OF ARRAY
0311- A0 04         1330        LDY #4        COMPUTE SIZE OF PREAMBLE
0313- B1 9B         1340        LDA (LOWTR),Y      # DIMENSIONS
0315- 0A            1350        ASL                *2, AND CLEAR CARRY
0316- 69 05         1360        ADC #5             +5 (2 FOR NAME)
0318- 65 9B         1370        ADC LOWTR             (2 FOR OFFSET)
031A- 48            1380        PHA                   (1 FOR # DIMS)
031B- A5 9C         1390        LDA LOWTR+1
031D- 69 00         1400        ADC #0             ADD CARRY
031F- 85 95         1410        STA ARYPT+1
                    1420 *---GET ENDING ADDRESS-----------
0321- 18            1430        CLC           ADD OFFSET TO GET ADDRESS OF END
0322- A0 02         1440        LDY #2
0324- B1 9B         1450        LDA (LOWTR),Y
0326- 65 9B         1460        ADC LOWTR
0328- 85 9D         1470        STA ARYEND
032A- C8            1480        INY
032B- B1 9B         1490        LDA (LOWTR),Y
032D- 65 9C         1500        ADC LOWTR+1
032F- 85 9E         1510        STA ARYEND+1
                    1520 *---SET UP POINTER TO START------
0331- 68            1530        PLA
0332- A8            1540        TAY
0333- A9 00         1550        LDA #0
0335- 85 94         1560        STA ARYPT
0337- A6 95         1570        LDX ARYPT+1
                    1580 *---LOOP TO SET ELEMENTS ZERO----
0339- 91 94         1590 .4     STA (ARYPT),Y
033B- C8            1600        INY
033C- D0 03         1610        BNE .5        ...USUALLY
033E- E8            1620        INX           ...NEXT PAGE
033F- 86 95         1630        STX ARYPT+1
0341- C4 9D         1640 .5     CPY ARYEND    AT END YET?
0343- D0 F4         1650        BNE .4        ...NO
0345- E4 9E         1660        CPX ARYEND+1
0347- D0 F0         1670        BNE .4        ...NO
                    1680 *---CHECK IF ANOTHER ARRAY-------
0349- 20 B7 00      1690        JSR CHRGOT
034C- D0 B9         1700        BNE .2        ...YES, UNLESS SYNTAX ERROR
034E- 60            1710        RTS
```

Turn an Index into a Mask.................Bob Sander-Cederlof

How do you write a program that will turn a number from 0 to 7 into a bit mask $01, $02, ...$40, $80? I want an index of 0 to return $01, 1 to return $02, 2 to return $04, and so on up to 7 returning $80.

The simplest, shortest, and speediest is to use a direct table look-up. Assuming the byte with the index value is in the A-register, the code would look like this:

```
       AND #7          isolate index bits
       TAX             index to X-register
       LDA TABLE,X     get mask from table
```

and the table would look like this:

```
       TABLE .HS 01020408
             .HS 10204080
```

This technique has the wonderful advantage that if you need a different translation, you can simply use a different table. For example, if you want the reverse pattern, with 0 returning $80 and 7 returning $01, simply change the table to:

```
       TABLE .HS 80402010
             .HS 08040201
```

The table lookup method has the shortest code, but counting the table does take 14 bytes. If you don't worry so much about speed and flexibility, you can write a little loop that will create the mask value like this:

```
       MAKE.MASK.2
               AND #7      isolate index bits
               TAX         index into X-register
               LDA #$01    initial mask value
       .1      ASL         shift loop to position
               DEX         to Xth bit
               BPL .1      shifts once to many
               ROR         restore after extra shift
               RTS
```

I put an RTS at the end because this piece of code makes a nice size subroutine. Nevertheless, for comparison to the table lookup code above, let's count neither the JSR to call it nor the RTS at the end. The shift-loop method takes only 10 bytes, four less than the table lookup. But it is slower, taking 14 cycles if the index is 0, 21 if 1, up to 63 for an index of 7. Sometimes saving four bytes is more important that speed, and sometimes speed is more important.

To generate the reverse sequence with the shift loop method, make three simple changes to MAKE.MASK.2: the initial mask value from $01 to $80; the ASL to LSR; and the ROR to ROL.

Note that both techniques shown above use the X-register. If the X-register is busy, you could use the Y-register instead.

Just for the challenge, I wanted to see if I could write a reasonably efficient index-to-mask routine that did not use the X- or Y- registers at all.

The first method that came to mind was fast enough, but took too much space and did not seem creative. It involved a series of CMP and BEQ instructions to branch to 8 different LDA's:

```
SILLY.WAY
        AND #7   isolate index
        BEQ .0   index=0
        CMP #1
        BEQ .1   index=1
        ...
        CMP #6
        BEQ .6     index=6
        LDA #$80   index=7
        RTS
.0      LDA #$01
        RTS
.1      LDA #$02
        RTS
        ...
.6      LDA #$40   index=6
        RTS
```

If I had written every line above, you would see that it takes 52 bytes.

Next I though of a more efficient way to do the CMP's so that not so many were needed.

```
NOT.SO.SILLY.WAY
        AND #7   isolate mask
        BEQ .0   index=0
        CMP #4
        BEQ .4   index=4
        BCS .60  index=5, 6, or 7
        CMP #2   index=1, 2, or 3
        BEQ .2   index=2
        BCS .3   index=3
        LDA #$02   index=1
        RTS
.60     CMP #6   index=5, 6, Or 7
        BEQ .6   index=6
        BCS .7   index=7
        LDA #$20   index=5
        RTS
.0      LDA #$01   index=0
        RTS
.2      LDA #$04   index=2
        RTS
    and so on.
```

This method takes a total of 46 bytes.

Here is one which is even shorter, which uses "tricky" arithmetic.

```
TRICKY.WAY
        AND #7
        CMP #2
        BCC .5   (0 or 1) plus 1
        BEQ .5   (2) plus CARRY plus 1 --> 4
        CMP #4
        BCC .4   (3) plus 4+1 --> 8
        BEQ .3   (4) plus 6+4+1+C --> $10
        CMP #6
        BCC .2   (5) plus $10+6+4+1
        BEQ .1   (6) plus $1E+$10+6+4+1+C
        ADC #$3F   (7) plus $3F+$1E+$10+6+4+1+C
.1      ADC #$1E
.2      ADC #$10
.3      ADC #6
.4      ADC #4
.5      ADC #1
        RTS
```

Not counting the RTS, that is 31 bytes. Cases 0 and 1 take only 9 cycles. The longest one, when the index is 7, takes 32 cycles.

All of these longer methods can be made to generate the reverse sequence by simply inverting the index before beginning the tests. Use "EOR #7" before the "AND #7".

I came up with an even trickier version, which shaved another byte or two off TRICKY.WAY. Believe it or not, it really works:

```
TRICKIER.WAY.REVERSE
        EOR #7
TRICKIER.WAY
        AND #7      isolate index
        SEC         00-01-02-03-04-05-06-07
        ROL         01-03-05-07-09-0B-0D-0F
        CMP #3
        BCC .0      turn 0 into $01
        CMP #7
        BCC .12     03-->02, 05-->04
        ADC #6      ..-..-..-0E-10-12-14-16
        CMP #$12
        BCC .34     0E-->08, 10-->10
        ADC #$2B    ..-..-..-..-..-3E-40-42
        CMP #$42
        BCC .56     3E-->20, 40-->40
        ASL         42-->84-->80
.56     AND #$E0
.34     AND #$F8
.12     AND #$FE
.0      RTS
```

If the index is 0, this one takes 11 cycles. Worst case is for index 7, at 34 cycles.

A source file on the quarterly disk will include all of the above examples, plus a driving program that runs through all 8 cases and displays the results for each and every method.

In real life, I would probably use the shift-loop or the table look up. Most likely the table lookup, because it is the easiest to understand and modify, and by far the shortest in time. Nevertheless, it is very useful to experiment with other techniques. You learn a lot from the experience, and it is fun!

Put Your Messages on the Screen..............William M. Reed

COUT is slow. COUT with DOS looking on is even slower. And I suppose with ProDOS, more so. If you want to get a short message on the screen in a hurry, you can bypass COUT and put it there directly.

In all of the following examples I am going to assume that the message is stored in RAM exactly as it should be on screen, and that after the last character is a byte with $00 in it. I also assume that you are only writing one line, so that the message will not spill over to another line.

Here is a loop that writes a message on the bottom line of the screen:

```
MESSAGE
       LDY #0
.1     LDA MESSAGE,Y
       BEQ .2           ...END OF MESSAGE
       STA $7D0,Y
       INY
       BNE .1           ...ALWAYS
.2     RTS
```

If you want to write on the current line, whose base address is kept by the monitor in BASL and BASH ($28 and $29), just change the STA $7D0,Y line to STA (BASL),Y.

All well and good for 40 columns, but what about the 80-column //e and //c screens? Well, you can still do it, like this:

```
MESSAGE.80
       LDX #0         MESSAGE INDEX
.1     TXA
       LSR            COLUMN/2, ODD/EVEN TO CARRY
       TAY            INDEX INTO SCREEN MEMORY
       LDA MESSAGE,X
       BEQ .3         ...END OF MESSAGE
       STA PAGE1
       BCS .2         ...ODD, PAGE 1
       STA PAGE2      ...EVEN, PAGE 2
.2     STA (BASL),Y
       INX
       BNE .1         ...ALWAYS
.3     RTS
PAGE1  .EQ $C054
PAGE2  .EQ $C055
```

Of course, these routines put the messages on the screen only. But that may be just what you want, to put messages on the screen without affecting the report going out to file or printer. Also, these routines do not handle CR, end of line, scroll, etc; but they sure get to the screen in a hurry!

Bibliography on Apple Hi-Res Graphics......Bob Sander-Cederlof

There has been a lot of material published in the last seven years about Apple's hi-res graphics. The problem is finding it! Most of the neat programs and explanations have not yet made it from the pages of various magazines into full size books. I recently decided to make a list, so that I don't have to keep leafing through mile-high stacks of magazines. Since I have never been a devotee of Pascal, I purposely omitted most articles relating to graphics in that language. I also omitted reviews and announcements of commercial hi-res products.

I looked through my book shelves and noted all books I could find there. I also went through all my back issues of Byte, Micro, Call APPLE, and Apple Orchard. Still to go are Nibble, Kilobaud, Softalk, and Creative Computing.

Books

Apple Graphics & Arcade Game Design, Jeffrey Stanton. The Book Co., 1982, 288 pages, $19.95. By the time you work through this one, you have a functioning hi-res arcade game!

Apple II Graphics, Ken Williams. Prentice Hall, 1983, 150 pages, $19.95. (Originally a series of articles in Softline Magazine, Sep 81 through Jan 83.)

Applied Apple Graphics, Pip Forer. Prentice-Hall, 1984, about 400 pages plus diskette, price unknown. Lo-res, Hi-res, 3-D, etc., with over 50 program in BASIC on disk.

Graphically Speaking, Mark Pelczarski. Softalk Books, 1984, 170 pages, $19.95. Originally a series of articles which ran from May 1982 through September 1983 in Softalk Magazine. Includes many programs in Applesoft and assembly language. Covers drawing, animation, filling, packing/unpacking, and 3-D. Disk available.

Microcomputer Graphics, Roy E. Myers. Addison-Wesley, 1982, 282 pages, $12.95. More than 80 Applesoft programs. 2-D and 3-D graphics, windowing, transformations, hidden lines, and much more. Disk available.

Books with some material on Apple Graphics

Animation, Games, and Sound for the Apple II/IIe, Tony Fabbri. Prentice-Hall, 144 pages.

Enhancing Your Apple II, Volume I, Don Lancaster. Howard Sams & Co., 1984, 268 pages, $15.95. Hardware and software tricks for switching between modes and screens dynamically, programs for hundreds of hi-res colors and patterns, fast screen fill. Good explanations of the way things work.

What's Where in the Apple, William F. Luebbert. Micro Ink, 1982, about 300 pages. First half of book is text describing Apple; chapter 14 covers lo-res graphics, and chapter 16 covers hi-res graphics. Includes details about hardware switches, memory mapping, and firmware.

Magazine columns

Assembly Lines, Roger Wagner, Softalk Magazine. From March 82 to June 83 this column covered various topics in Apple hi-res graphics. It should be made into a book, but has not yet been.

The Graphics Page, Bill Budge, Softalk Magazine. Oct 83 through Jun 84. Deep material, by the author of Pinball Construction Set. Further installments were promised, but not yet seen.

Apple II Graphics, Ken Williams, Softline Magazine. Sep 81 through Jan 83. Now available in book form (see above).

Graphically Speaking, Mark Pelczarski, Softalk Magazine. May 82 through Sep 83. Now available in book form.

Magazine Articles

Byte

Apple FAX: Weather Maps on a Video Screen, Keith H. Sueker. Jun 84, 146-151.

CHEDIT: a Graphics-Character Editor, Jerry Sweet. May 82, p426-444.

Double the Apple II's Color Choices, Robert H. Sturges. Nov 83, p449-463.

Double-Width Silentype Graphics for Apple, Charles Putney. Feb 82, p413-423.

GRPRINT: an Apple Utility Program, Douglas Arnott. Dec 82, 398-403.

Interactive 3-D Graphics for Apple II, Andrew Pickholtz. Nov 82, 474-505.

Kinetic String Art for the Apple, Louis Cesa. Nov 80, p62-63.

More Colors for your Apple, Allen Watson, Steve Wozniak. Jun 79, p60-68.

New Shape Subroutine for the Apple, Richard T. Simoni. Aug 83, p292-309.

Picture Perfect Apple, Phil Roybal. Jan 81, p226-235.

Shape Table Conversion for the Apple II, Dave Partyka. Nov 79, p63.

Simplified Theory of Video Graphics, Allen Watson. Part 1, Nov 80, p180-189. Part 2, Dec 80, p142-156.

Three-dimensional Graphics for the Apple II, Dan Sokol, Nov 80, p148-154.

Micro

Apple Bits, Richard C. Vile Jr. Part I, Sep 81, p75-77. Part 2, Oct 81, p94-96. Part 3, Nov 81, p105-108. (Lo-Res)

Apple Color Filter, Stephen R. Berggren. Jun 81, p53-54.

A Hi-Res Graph Plotting Subroutine in Integer BASIC for the Apple II, Richard Fam. Feb 80, p9-10.

Apple Graphics, staff. Sep 81, p49. Intro to several other articles.

Apple Graphics for Okidata Microline 80, Gary Little. May 83, p80-86.

Apple Hi-Res Graphics and Memory Use, Dan Weston. Nov 82, p79-81.

Apple II High Resolution Graphics Memory Organization, Andrew H. Eliason. Oct-Nov 1978, p43-44.

Apple II Hi-Res Picture Compression, Bob Bishop. Nov 79, p17-24.

Apple Pascal Hi-Res Screen Dump, Robert D. Walker. Feb 83, p54-55.

Apple Shootdown, a lo-res graphics game, Eric Grammer. Nov 82, p72-73.

A Versatile Hi-Res Graphics Routine for the Apple, Adam P. King. Mar 83, p77-81.

Constructing Truly 3-D Mazes, Dr. Alan Stankiewicz. Aug 84, p19-21.

Creating Shape Tables, Improved!, Peter A. Cook. Sep 80, p7-12.

Define Hi-Res Characters for the Apple II, Robert F. Zant. Aug 79, p44-45.

Getting Around the Apple Hi-Res Graphics Page, Eagle Berns. Nov 82, p93-95.

Graphing Rational Functions, Ron Carlson. Dec 80, 7-9.

Hi-Res Characters for Logo, Dan Weston. Sep 83, p50-53.

Hi-Res Screen Dump for Epson MX-80, Robert D. Walker. Apr/May 84, p55-61.

How to Do a Shape Table Easily and Correctly, John Figueras. Dec 79, p11-22.

Introduction to 3-D Rotation on the Apple, Chris Williams. Nov 82, p99-101.

Paddle Hi-Res Graphics, Kim G. Woodward. Sep 81, p68-69.

Random Number Generator in Machine Language for the Apple, Arthur Matheny. Includes a graphics simulation of a globular cluster. Aug 82, p57-60.

SHAPER: A Utility Program for Managing Shape Tables, Clement D. Osborne. Sep 81, p50-56.

Sun and Moon on the Apple, Svend Ostrup. Jan 83, p35-37. Hi-res simulation of orbits and phases.

True 3-D Images on Apple II, Art Radcliffe. Sep 81, p71-73.

Call-A.P.P.L.E.

80-column //e Lo-Res Graphics, Rob Moore. Jul 83, p9-13.

Adding XPLOT to Applesoft, Mark Harris. Apr 84, p17-18,24.

A Higher Text Apple-cation, Donal Buchanan. Nov 82, p47-50. Using Higher Text for ancient alphabets.

Animation with Data Arrays, Pat Connelly. Nov-Dec 80, p11-17.

Apple Gaming: Playing Card Generation, Jim Hilger. Nov-Dec 79, p39-45; Jan 80, p39. Hi-res playing card pictures from Integer BASIC.

Applesoft Firmware Card Hi-Res Routines, Steve Alex. Oct 79, p33.

Applesoft Graphics Mover, Homer O. Porter. Sep 83, p29-31.

Arcade Graphics Techniques, Chris Jochumson. Apr 83, p9-14.

Character Generator ROM, Ian M. Jackson. Nov 82, p21-29. Programs for moving a Higher Text font into ROM.

Color 21, Darrell Aldrich. Jul-Aug 79, p21.

Color Me Apple, M. A. Iannce. Nov 82, p9-18. In-depth explanation of hi-res color with demo program.

Doing the Splits, Roy Myers. Aug 82, p61-65. Making room for hi-res pictures by moving your program.

Graphic Garbage Collection, Richard Cornelius & Melvin Zandler. Nov 82, p53-55. Lets you watch garbage collection activity on the hi-res screen.

Higher Text in Action, Steve Brugger. Jan 84, p30-31.

Higher Text on the Loose, Val J. Golding. Jun 81, p47-49. Explanation of the background functions of Higher Text.

Hi-Res Dump program modification, Tom Lewellen. Jul-Aug 79, p36.

Hi-Res Full Scroll, Edward C. So. Feb 82, p23-34. Scroll up, down, left, or right by one pixel position at a time.

Hi-Res Hi-Jinx, Edward C. So. Apr 82, p59. POKEing and PEEKing dots.

Hi-Res Screen Switch (program), Wes Huntress. Jul-Aug 80, p48-49.

Hi-Res Slide Show, Stowe Keller. Dec 83, p49-54.

Magic Square Dance, J. Taylor. Sep 83, p51-52.

MX-100 Hi-Res Dump, Bruce C. Detterich. Mar 82, p41-49.

Painting the RAMcard, Donald W. Miller Jr. Apr 83, p51-54.

Picture Compression, Edward C. So. May 82, p21-35. Very complete, builds on Bob Bishop's attempts.

Playing Card Generator, Vincent Aderente. Nov 82, p31-35. Applesoft versions of Jim Hilger's stuff.

Scrunch, Darrell Aldrich. Jun 79, p21-23. Squeeze four pictures into one screen.

Shape Display Utility, Major Peter M. Beck. Mar-Apr 80, p39.

Shape Table Splicer, Cyrus W. Roton. Sep 83, p33-35.

Slow Plot, Jim Morriset. Nov 82, p63-64. Speed control for hi-res drawing.

Smooth Animation, Jonathan Kandell. Feb 83, p61-62.

The Graphics Toolkit, Randi J. Rost. Part 1: Apr 84, p10-15 (screen mapping). Part 2: May 84, p23-26. Part 3: Aug 84, p43-48. Line drawing algorithms, disassembly of Applesoft HPLOT.

Three Dee Demos, David Sun. Jan 83, p49-51.

Understanding Hi-Res Graphics, Loy Spurlock. Jan 80, p6-15.

Using the Splitter, Norman L. Kushnick. Jan 83, p53-55. More help in making room for pictures.

Why Don't You Watch Where You're Going?, Kenneth Manly. Oct 80, p25-28. A hi-res SCRN function.

Zoom, Neil Konzen. Jan 80, p28-32. Expand a 1/9th screen to full size.

Apple Orchard

Double-Size Graphics for the Silentype, Bruce F. Field. Spring 81, p30-34.

Hi-Res Dump Routine for Integral Data Printer (IDS-225), Darrell & Ron Aldrich. Mar-Apr 80, p54-55. (Originally published in Call APPLE).

Hi-Res Graphics: Resolving the Resolution Myth, Bob Bishop. Fall 80, p7-10.

How the Dot Patterns Produce Colors, Allen Watson III. Jan 84, p44-46.

How the Double Hi-Res Hardware Came to Be, Allen Watson III. Jan 84, p42,43.

Notes on Hi-Res Graphics Routines in Applesoft, C. K. Mesztenyi. Spring 81, p17-19.

Practical Super Hi-Res Graphics, R. H. Good. Spring 81, p20.

Secrets of Professional Graphics, William Harvey. Part I, Behind the Scenes, Sep-Oct 82, p64-72. Part II, The Real Challenge: Putting It All Together, Nov-Dec 82, p36-53. Part III, Techniques of Animation, Mar 83, p62-69.

Shape Definition Conversion Table, David G. Huffman. Fall 81, p78-79.

Shaping Up with the Apple II, Mark L. Crosby. Mar-Apr 80, p37-45.

The Mysterious Orange Vertical Line, Pete Rowe. Fall 80, p11.

True 16-color Hi-Res, Allen Watson III. Jan 84, p26-41.

Understanding Hi-Res Graphics, and how to include text in your Hi-Res Graphics Programs, Loy Spurlock. Fall 80, p12-21.

Some Great New Books.......................Bob Sander-Cederlof

"Beneath Apple ProDOS", by Don Worth and Pieter Lechner. Quality Software, 1984, 276 pages plus 10 page reference card, $19.95. (By it from us for $18 plus shipping.)

We have been waiting a long time for this one, by the authors of "Beneath Apple DOS". If you read that one, you'll want this one too. And if you use or plan to use ProDOS, you will almost REQUIRE this new book.

Apple has documented ProDOS pretty thoroughly, but just TRY to get a copy of their books. Hardly any Apple dealers stock the reference manuals now. Apple requires a minimum order to buy the manuals, and they are a relatively slow moving item. Hence, dealers don't order them. Some we have talked with lately refused to admit they knew of the existence of even the Apple //e Reference Manual (over 18 months old now)! And Apple so far will not sell the books to anyone who is not an authorized Apple dealer. Catch-22, right?

But even if you have Apple's ProDOS reference manuals, as I do, you still need "Beneath Apple ProDOS". Look at the table of contents, and see if you can resist.

The most heavily thumbed pages in my copy of "Beneath Apple DOS" are the ones which give detailed comments on the entire DOS assembly language image. Unfortunately, the equivalent section does not come bound in to "Beneath Apple ProDOS". Since Apple has decided to freeze DOS, a published commentary is possible. But ProDOS is deliberately kept warm and fluid. So far there are at least four versions around; all have the same characteristics and machine language interface, but subroutines have been shuffled and rewritten. A line-by-line commentary could become obsolete every six months.

A special coupon is bound into the book at the place where you would expect the commentary. If you want the commentary, you remove the coupon page, fill in your name, address, and ProDOS version number, and send it with $12.50 to Quality Software. With the commentary you will receive a new coupon so your can order a subsequent supplement when ProDOS changes versions.

"Assembly Cookbook for the Apple II/IIe", Don Lancaster. Howard Sams & Co., 1984, 408 pages, $21.95. (Buy it from us for $20 plus shipping.)

Don is sold on the synergistic combination of a full-screen 80-column word processor for handling source code with an assembler. His favorite pairing is Applewriter //e with EDASM (from DOS ToolKit). Consequently a large section of the book is devoted to how the marriage is performed, what the advantages are, and how to work around or ignore the disadvantages. Don knows Applewriter inside out, and uses it for all his word processing as well as for programming. There are some distinct advantages to using the same editor for both: writing books about assembly language programming is easier; only one set of

commands, tricks, and quirks need be learned. Applewriter //e's WPL language helps overcome the disadvantages of using a screen-oriented processor on line-oriented information.

The second half of the book contains sample assembly language programs, explained in detail. These are not your run-of-the-mill examples, but great subroutines and programs you can actually use, as well as learn from.

"Microcomputer Design and Troubleshooting", by Eugene M. Zumchak. Howard Sams & Co., 1982, 350 pages, $17.95. (Buy it from us for $17 plus shipping.)

From time to time I am called upon to understand and work with electronics. My degree is in Electronic Engineering, but I got it in the vacuum tube era (over 20 years ago). What now fits on one chip used to fill a whole ship.... Anyway, I struggle through. But I have found a book recently that has really helped: it is not really a new book, but is new to me.

Gene Zumchak has a unique approach, which is PRACTICAL. He believes in designs which are easy to troubleshoot. He tells how adding a few low cost components here and there will avoid the expense of a logic analyzer and three weeks of debugging time. For example, using an EPROM emulator and a few LED's in critical places in a microprocessor design could save endless hours of burning and erasing EPROMs, attaching logic analyzer leads and watching oscilloscope traces, and pulling all your hair out. Although every chapter has helpful ideas in the areas of trouble prevention and diagnosis, chapter 6 is devoted entirely to the subject. Another feature Gene promotes is low power consumption.

Jack Lewis is president of Micromation, a company which makes hardware for use with the Hero-1 Robot. They have designed interfaces between Apple and Hero, speech input processors, and much more. When Jack began, he contracted with Gene Zumchak to teach his people the techniques which are now in this book. Jack is the one who recommended the book to me.

And now I recommend the book to you, if you like to dabble in hardware design. Even practicing designers will find the ideas well worth the price of reading the book.

I also recommend "The Computer Journal", a monthly newsletter/magazine published by Art Carlson. $24/year (U.S.) gets you regular articles such as "Build a 68008 CPU Board for the S-100 Bus", "Electronic Dial Indicator", "Writing Your Own Threaded Language", and "Interfacing Tips and Troubles". Write to Art at P. O. Box 1697, Kalispell, MT 59903.

Beneath Apple ProDOS

CONTENTS

These items are also available from S-C Software.

quikLoader -- $170   D MAnual Controller -- $85.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).